**Poniższy artykuł dedykujemy szczególnemu kręgowi odbiorców: „profesjonalnym hobbystom". Tym razem gratka dla tych, którzy mozolnie próbują unowocześnić nasz przemysł wprowadzając mikroprocesorowe sterowniki.**

**Bezspornie najbardziej efektywną obsługę sterowników mikroprocesorowych zapewnia oprogramowanie pisane w języku ASSEMBLER. Trzeba jednak uwzględnić fakt, że współpraca specjalistów różnych dziedzin z programistami przebiega dość opornie. Optymalnym wyjściem byłoby tworzenie oprogramowania przez tych, którzy najlepiej rozumieją istotę procesów kontrolowanych przez komputer. Taką szansę stwarza jeden z najłatwiejszych do opanowania języków — BASIC. Informacja o dostępnej wersji tego języka, umożliwiającej sterowanie w czasie rzeczywistym (i to pracującej pod system CP/M!) może okazać się przełomowa dla całej gamy nowych zastosowań mikroprocesorów do sterowania.**

# INDUSTRIAL REAL-TIME BASIC – dla mikrokomputerów

INDUSTRIAL REAL-TIME BASIC (IRTB) został zdefiniowany jako moduł stanowiący rozszerzenie klasycznego języka Basic. IRTB jest przeznaczony do wykorzystania w sterowaniu automatyzacją procesów technologicznych i w monitorowaniu otoczenia komputera, czyli w zastosowaniach, które można rozdzielić na określoną liczbę współbieżnych procesów — w dużym stopniu niezależnych i asynchronicznych, ale okazjonalnie się z sobą komunikujących i synchronizujących. Program dla takich zastosowań jest więc z natury rzeczy „wielowątkowy" i musi mieć na ogół możliwość działania w nieograniczonym horyzoncie czasowym.

Język BASIC był niedawno prezentowany na łamach INFORMATYKI [1] i w związku z tym niniejszy artykuł koncentrować się będzie na zagadnieniach związanych z implementacją IRTB.

IRTB zawdzięcza swoją atrakcyjność prostocie koncepcyjnej zastosowanych mechanizmów oraz... brakowi liczącej się konkurencji — oczywiście w klasie prostych zastosowań przemysłowych i laboratoryjnych. Pewną rolę grać też będzie niewątpliwie przyzwyczajenie użytkowników (szczególnie nie będących informatykami), którzy dotychczas skazani byli na programowanie w językach asemblerowych lub stosowali jedną z wielu wersji języka BASIC, adaptowanych do celów sterowania [1]). Obecnie odmiany IRTB stanowią podstawowe oprogramowanie kilku niedawno wprowadzonych na rynki zachodnie uniwersalnych sterowników mikrokomputerowych. Do dziś nie powstała jednak na świecie pełna (zgodna z normą) implementacja IRTB.

Zalety IRTB w oprogramowywaniu układów sterowania były inspiracją podjęcia w PPZ COMPUTEX pracy nad własną implementacją tego języka. Pilotująca implementacja została dokonana na systemie CS-80 i nosi nazwę INDUSTRIAL BASIC.

Przewidywany nakład pracy oraz zapewnienie możliwości przenoszenia na inne systemy były decydującymi czynnikami w wyborze sposobu implementacji. Zdecydowano się na interpreter, mimo że powoduje to znaczne spowolnienie wykonywania programów użytkowych. W systemie z założenia pracującym w czasie rzeczywistym może się to okazać istotnym ograniczeniem. W zrealizowanej implementacji gra ono rolę dopiero przy potrzebie reakcji w czasie krótszym od 0,1—0,2 s.

W chwili rozpoczęcia prac opierano się na propozycji standardu IRTB z września 1981 [2]. Późniejsze poprawki (do rozdz. 5.2 i 5.6 tego dokumentu) nie miały wpływu na przygotowywany program interpretera, ponieważ dotyczyły nieimplementowanych elementów języka: deklaracji struktur danych oraz instrukcji niedeterministycznego oczekiwania zadania na komunikację z innym zadaniem (blok SELECT). Ponadto, syntaktyka języka została w kilku miejscach w niewielkim stopniu zmieniona, co podyktowane było względami implementacji. Język rozbudowano też o instrukcje spoza standardu, przeznaczone głównie do współpracy z zasobami mikrokomputera (CLOCK, PEEK, POKE, wywołanie podprogramu w kodzie maszynowym).

Program INDUSTRIAL BASIC może być wykorzystywany w dwóch wersjach:

— uruchomieniowej, jako program biblioteczny dyskowego systemu operacyjnego MAGOS (kompatybilnego z CP/M)

— rezydencyjnej, umieszczanej w pamięci EPROM sterownika mikrokomputerowego.

Pierwsza wersja stosowana jest na etapie projektowania i testowania programów w warunkach laboratoryjnych,

[1]) W Polsce był to przede wszystkim implementowany przeze mnie wielozadaniowy BASIC RT, oferowany w różnych wariantach przez ISEP PW, PPZ COMPUTEX, a ostatnio także MIKRONIKĘ z Poznania.

## Na Węgrzech można...

Dla wielu rodaków Węgry to kraj taniego wina i sweterków, za którymi uganiają się polskie turystki. Jest to jednak również kraj, w którym zaskakująco wiele (a realia w końcu są niemal identyczne) dzieje się w mikroinformatyce. Zacznijmy od rzeczy nam najbliższej: w nakładzie 30 tys. egzemplarzy ukazuje się tam dwumiesięcznik „Mikro" — μ-MIKROSZAMITOGEP-MAGAZIN (biorąc pod uwagę liczbę ludności, odpowiadało by to u nas 100 tys. nakładu). Pismo nie tylko dobrze się sprzedaje, ale również przynosi dochód — pokaźna jego część pochodzi z ogłoszeń. Okazuje się bowiem, że na Węgrzech opłaca się np. reklamować... COMMODORE C64. Wprawdzie kompletny system (komputer, monitor, napęd dysków elastycznych) kosztuje ok. 70 tys. forintów (prawie trzyletnia przeciętna płaca), ale i nasza cena wolnorynkowa za kompletny system kształtuje się w takich granicach. Tyle że na Węgrzech można C64 kupić w sklepie. Za dwumiesięczne wynagrodzenie (10 tys. forintów) można natomiast kupić prosty mikrokomputer węgierskiej produkcji. Dokładnie odwrotnie niż u nas (cena MERITUM odpowiada obecnie wolnorynkowej cenie C64). Wracając jeszcze do ogłoszeń: zamieszczają je również producenci sprzętu profesjonalnego — widocznie nie mogą liczyć (odwrotnie niż nasi), że cokolwiek zrobią to i tak rozejdzie się spod lady.

Na tym jednak nie koniec rewelacji z węgierskiego rynku. Otóż na Węgrzech można także nabyć zachodnie profesjonalne mikrokomputery. W tym nie tylko IBM PC ale nawet... MACINTOSHA. Teraz to już jasne jakim cudem Węgrzy oferowali na Zachodzie oprogramowanie graficzne do tego mikrokomputera.

Mikrokomputerowa oferta na węgierskim rynku tylko częściowo finansowana jest za „państwowe" dolary. Tak jak u nas niektórzy dorabiają na sprowadzaniu z Zachodu końcówek do długopisów — na Węgrzech można dorobić... sprowadzając mikrokomputer. Cała operacja jest zupełnie legalna i co najważniejsze — istnieją precyzyjne przepisy określające opodatkowanie (ustalona jest jednoznaczna kwota cła dla komputerów przeznaczonych do odsprzedania — i na tym koniec). W ten prosty sposób państwo nie tylko nie dokłada do upowszechniania mikroinformatyki, lecz wprost przeciwnie — ciągnie z tego określone zyski. Na pośrednictwie korzysta również nabywca, który dostaje komputer przetestowany i... z gwarancją. Czy to możliwe abyśmy mogli w naszym kraju zastosować równie prostą metodę?

Wracając jeszcze do piśmiennictwa mikroinformatycznego na Węgrzech. Przemiennie z dwumiesięcznikiem „Mikro" mają ukazywać się monotematyczne broszury (tu kolejna różnica: mikroKLAN próbuje wydawać broszury — natomiast Węgrzy już je drukują). Przygotowywana jest broszura opisująca sposób samodzielnego złożenia mikrokomputera, inna z kolei poświęcona zostanie językowi LOGO. Zestawiając tematy okazuje się, że wiele pomysłów jest niemal identycznych. Jednak — z relacji węgierskich kolegów — ich droga od pomysłu do realizacji, choć również nie jest usłana różami, wygląda na znacznie krótszą. I jeszcze jedno: honorarium, jakie można otrzymać na Węgrzech za dobry tekst, zbliżone jest do miesięcznej pensji inżyniera. Takie podejście daje możliwość „pozyskania" naprawdę dobrych autorów. W konsekwencji, mimo umowy z zachodnioniemieckim pismem CHIP, umożliwiającej przedruk dowolnych publikowanych tam materiałów, Węgrzy jeszcze ani razu nie potrzebowali z tej możliwości korzystać.

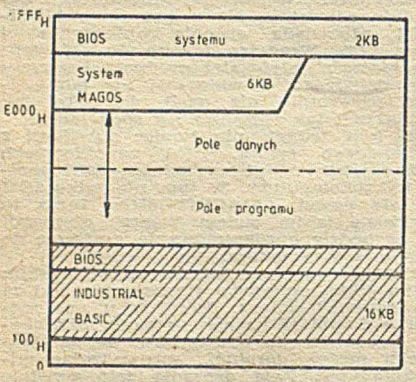
Do tej pory zwolennicy mikroinformatyki z zazdrością patrzyli na kraje zachodnie. Okazuje się, że możemy również zazdrościć krajom o tym samym systemie. Tylko na czym to polega, że tam można, a u nas niezupełnie?

AJP

docelowo eksploatowanych już na rzeczywistym obiekcie przez wersję drugą. Takie podejście skraca czas przygotowywania programów.

INDUSTRIAL BASIC składa się z następujących modułów:
— edytor programów użytkowych (tylko w wersji uruchomieniowej)
— szybki interpreter
— pakiet arytmetyki zmiennoprzecinkowej
— biblioteka funkcji matematycznych (opcja)
— system operacyjny czasu rzeczywistego
— moduł inicjacji
— moduł BIOS.

Moduł BIOS (ang. Basic Input Output System) umożliwia szybkie przystosowanie programu do praktycznie dowolnego mikrokomputera wykorzystującego procesor Z80. Umieszczona na początku modułu tablica skoków powoduje, że przeniesienie programu na inny sprzęt wymaga jedynie napisania nowych procedur współpracy z nowymi zasobami sprzętowymi.




**Rys. 1. Mapa pamięci komputera**

Użytkownik ma do swojej dyspozycji — dla programu w języku BASIC i jego dynamicznych struktur danych — do 45 KB pamięci. Podział pamięci na pole programu i pole danych jest płynny, jednak program użytkowy nie może nałożyć się na system operacyjny wymagany dla odczytu-zapisu programu na dyskietce.

Program użytkownika składa się z dwóch części: segmentu deklaracji globalnych i segmentu zadań współbieżnych. Deklaracje wprowadzają do programu nazwy obiektów globalnych:
— „wiadomości" wymienianych między zadaniami o treści numerycznej i określonej długości (deklaracja MESSAGE)
— sygnałów generowanych programowo lub przerwaniem zewnętrznym (deklaracja PROCESS EVENT)
— portów we-wy cyfrowego i analogowego (deklaracja PROCESS INPUT/OUTPUT)
— zmiennych wspólnych dla wszystkich zadań (deklaracja SHARED).

Segment zadań współbieżnych obejmuje zbiór sekwencyjnie definiowanych programów (zadań), ujętych w nawiasy syntaktyczne PARACT (ang. PARallel ACTivity) i END. Wielkość pola danych, koniecznego dla zadania, może być za każdym razem dowolnie deklarowana na podstawie oszacowania dokonanego przez programistę.

Z chwilą podania komendy RUN następuje wejście w moduł inicjacji, który między innymi przydziela bufory związane z zadeklarowanymi obiektami globalnymi. Sprawdzana jest też poprawność budowy programu. Pomyślne zakończenie tych operacji sygnalizowane jest komunikatem:

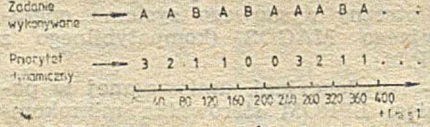
**Preprocessing completed, time xx:xx:xx**

W tym momencie rozpoczyna swój bieg zadanie współbieżne, które umieszczone jest jako syntaktycznie pierwsze w programie. Wszystkie inne zadania — zgodnie z wymaganiem standardu — muszą być uruchomione jawnie za pomocą instrukcji START.

**Opis implementacji**

Kluczowy problem — obsługę zadań współbieżnych — rozwiązano metodą dzielenia czasu procesora na odcinki 40 ms, przyznawane poszczególnym zadaniom przez system operacyjny. Implementowano pięć poziomów priorytetu, każdy dysponujący własną kolejką zadań oczekujących na wykonanie. Przyjęto zasadę, że zadanie z kolejki o niższym priorytecie może otrzymać procesor tylko wtedy, gdy kolejki wyższych priorytetów są puste. Aby uniknąć monopolizowania komputera przez zadania o wysokim priorytecie, przyjęto następujący algorytm postępowania: w chwili zwalniania procesora zadanie nie jest wstawiane na koniec kolejki „swojego" priorytetu, lecz do kolejki priorytetu bezpośrednio niższego niż ten, z którym było uprzednio wykonywane. W ten sposób w kolejnych aktywnych odcinkach czasowych dynamiczny priorytet zadania maleje do poziomu najniższego, po czym cykl powtarza się, począwszy od priorytetu deklarowanego w programie (rys. 2). Każde zawieszone zadanie i uaktywniane przez określone zdarzenie w systemie podejmuje bieg z priorytetem określonym w deklaracji — gwarantuje to szybką reakcję na sygnały ze sterowanego obiektu.

Dostęp zadań współbieżnych do wspólnych zasobów komputera — klawiatury operatora i ekranu — rozwiązano przez wzajemne wykluczanie się dostępu. Rozpoznanie w programie użytkowym instrukcji PRINT albo INPUT łączy się testowaniem znacznika




**Rys. 2. Przykładowy podział czasu procesora (zadanie A — priorytet deklarowany 3; zadanie B — priorytet deklarowany 1, priorytet 0 najniższy)**

dostępu do konsoli operatorskiej i w przypadku gdy jest ona zajęta, zawiesza zadanie w kolejce typu FIFO. Zawartość tej kolejki jest testowana z chwilą zakończenia realizacji każdej instrukcji PRINT lub INPUT. Jeżeli nie jest ona pusta, to uaktywnione jest kolejne z oczekujących zadań, a znacznik dostępu do konsoli pozostaje w stanie „zajęta".

Aby zidentyfikować zadanie, z którego wyświetlane na ekranie informacje pochodzą, ciąg alfanumeryczny poprzedzony jest zawsze syntaktycznym numerem zadania. Ponadto, w celu uniknięcia blokowania procesora przez oczekiwanie aktywnego zadania na wprowadzenie znaku z klawiatury, system operacyjny wykonuje test gotowości klawiatury przy każdej operacji szeregowania zadań. Pozytywny wynik testu powoduje wczytanie znaku do bufora i w przypadku, gdy jest to CR — wstawienie zadania zawieszonego na instrukcji INPUT do kolejki zadań oczekujących na procesor

Rozkazy komunikacji międzyzadaniowej (SIGNAL, PUT, GET, SEND, RECEIVE) wymagały dynamicznego tworzenia kolejek dostępu na podstawie deklaracji obiektów globalnych. W przypadku instrukcji SEND i RECEIVE, które realizują komunikację i synchronizację między zadaniami metodą tzw. rendez-vous (znana z języka ADA), są to dwie kolejki: zadań oczekujących na odebranie wysłanej przez nie wiadomości oraz zadań oczekujących na odebranie wysłanej wiadowości. Czas oczekiwania może być ograniczany klauzulą TIMEOUT. Drugą opcję powyższych instrukcji (nie definiowaną przez standard), stanowi ELSE. Jej zadaniem jest zapewnienie innej możliwości biegu zadania, gdy „rendez-vous" nie jest natychmiast możliwe. Pozwala to stworzyć m.in. programową konstrukcję zastępującą nieimplementowany blok SELECT:

```
100 DO
105 RECEIVE FROM „ALFA"
    TO X ELSE GOTO 125
    .....
120 EXIT DO
125 RECEIVE FROM „BETA"
    TO Y ELSE GOTO 145
    .....
140 EXIT DO
145 LOOP
```

Implementowany algorytm dopuszcza jednoczesny odczyt tych samych zmiennych przez kilka zadań (instrukcja GET), jednak pojawienie się żądania zapisu (instrukcja PUT) blokuje następne odczyty i zawiesza odpowiednie zadania do czasu zakończenia zapisu.

Zdarzenia, na które oczekują zadania wykonując instrukcję WAIT EVENT, są generowane programowo instrukcją SIGNAL lub przerwaniem zewnętrznym. Powoduje to wstawienie zadania do kolejki oczekiwania na procesor. Efektywna obsługa przerwania zewnętrznego nie jest więc natychmiastowa, lecz przesunięta w czasie, stosownie do priorytetu zadania obsługującego zdarzenie oraz aktualnego obciążenia procesora.

Opisana implementacja IRTB całkowicie spełniła pokładane w niej nadzieje. Narzuty czasowe związane z pracą systemu operacyjnego czasu rzeczywistego wynoszą średnio 5% cza-

su procesora, a stosowanie bardziej rozbudowanych trybów adresacji (m. in. adresacji indeksowej zmiennych wewnętrznych interpretera w bloku operacyjnym każdego zadania) nie spowodowało odczuwalnego spowolnienia interpretacji. Co więcej, badania porównawcze z interpreterm języka BASIC firmy MICROSOFT wykazały zbliżoną szybkość wykonywania programów testowych.

W najbliższym czasie przewidziana jest rozbudowa programu o obsługę sytuacji wyjątkowych (ang. exception). Pozwoli to na zachowanie kontroli nad programem nawet po ich wystąpieniu, w tym również po wystąpieniu błędów spowodowanych współpracą z otoczeniem, a nie wadami programu. Uczyni to z języka INDUSTRIAL BASIC program nadający się do sterowania procesów o szczególnie dużej odpowiedzialności.

**WOJCIECH WARSKI**
PPZ COMPUTEX, Warszawa

LITERATURA

[1] Iszkowski W., Maniecki M.: Standaryzacja języka BASIC. Informatyka 5—6/1983
[2] IRTB Draft Standard. Raport EWICS TC2 81/3.

**Wymiana programów jest dla wielu osób jedynym sposobem zgromadzenia biblioteki. Przekopiowanie programu dla siebie lub w przypadku wymiany z kolegą nie jest niczym zdrożnym. Zresztą, takie wydaje się być też zdanie opinii publicznej na Zachodzie, gdzie problem programowego piractwa jest szeroko dyskutowany. Inaczej jest w przypadku, gdy kopiowanie programów staje się dla kogoś źródłem dochodów. To już jest żerowanie na pracy innych, wyraźnie rozmijające się z zasadami etyki.**

# Tajemnice programowych piratów – ZX SPECTRUM

Kiedy w 1982 roku zaczęły pojawiać się pierwsze firmowe programy dla ZX SPECTRUM, producenci nie stosowali żadnych zabezpieczeń czy utrudnień w kopiowaniu programów. Programy były zapisywane na taśmie w sposób standardowy — z wykorzystaniem instrukcji SAVE. Aby przekopiować program, wystarczyło odczytać poprzedzający każdy blok nagłówek (17 bajtów zapisanych zaraz za odcinkiem charakterystycznego gwizdu). W nagłówku zawarta jest informacja pozwalająca komputerowi określić, czy następujący po nim blok zawiera program napisany w języku BASIC czy też w kodzie maszynowym, gdzie należy go załadować, czy po załadowaniu program powinien samoczynnie startować itp.

Wydzielenie i odczytanie nagłówka nie jest specjalnie skomplikowane; można się w tym celu posłużyć procedurami zawartymi w pamięci stałej ROM. Procedury takie można odnaleźć posługując się [1]. Jeżeli instrukcją BREAK przerwiemy realizację fragmentu programu napisanego w języku BASIC, to z łatwością można przepisać go ponownie na taśmę magnetofonową (przy zapisie należy pamiętać o podaniu numeru linii, od której rozpoczyna się samoczynna realizacja programu). Pozostałe bloki kopiujemy wykorzystując informację zawartą w nagłówkach (adres początkowy, długość bloku itp.).

Pierwszym utrudnieniem, które zaczęli stosować producenci oprogramowania, było tworzenie „nieprzerywalnych" programów. Program taki można zrealizować wykorzystując zmienną systemową DF-SZ. Określa ona, ile linii u dołu ekranu powinno zostać zarezerwowanych dla wyświetlania komunikatów. Standardowo zmienna ta ma wartość 2 (dwie linie) przechowywaną pod adresem $5C6B_H$ (23659 dziesiętnie). Jeżeli więc wykonana zostanie instrukcja:

**POKE 23659,0**

to niemożliwe jest wyświetlenie raportu o przerwaniu programu i w rezultacie dalsza praca komputera (trzeba go wyzerować wyłączając zasilanie).

Inny sposób polega na wykorzystaniu 2-bajtowej zmiennej ERR-SP. Jej wartość określa adres początkowy procedury obsługi błędu lub sytuacji specjalnych, takich jak właśnie polecenie BREAK. Jeżeli wartość zmiennej będzie wskazywała na procedurę odmienną od standardowej, to wykonana zostanie właśnie ta procedura (może ona np. niszczyć program). Wartość zmiennej ERR-SP przechowywana jest w dwóch kolejnych lokacjach pamięci o adresach $5C3D_H$ i $5C3E_H$ (dziesiętnie: 23613 i 23614). Żądaną wartość zmiennej ERR-SP można ustawić wykonując dwie instrukcje POKE.

Okazuje się, że programy „nieprzerywalne" można złamać w banalny sposób. Otóż zamiast ładować je instrukcją LOAD, która powoduje natychmiastowy start programu po wczytaniu go przez komputer — należy zastosować instrukcję MERGE. Wczytany w ten sposób program potulnie czeka w pamięci, aż ktoś go wylistuje.

Nieco lepiej zabezpieczone są programy, w których po głównym bloku zawierającym kod maszynowy występują króciutkie bloki nakładek. Wydawałoby się, że dysponując wiedzą o nagłówkach bez problemu ładujmy nakładki. Tymczasem okazuje się, że program odmawia prawidłowej pracy.

Nakładki to zazwyczaj krótkie programy, które wprowadzają pewne modyfikacje kodu, zawartego w głównym bloku. Jednak istota utrudnienia leży w czym innym. ZX SPECTRUM ma specjalne lokacje pamięci (3 bajty), których zawartość jest inkrementowana (zwiększa o jeden) pięćdziesiąt razy na sekundę. Ponieważ jest to częstotliwość ramki obrazu telewizyjnego, zmienną, której wartość przechowywana jest we wspomnianych bajtach nazwano FRAMES (ang. ramki). Zazwyczaj jedna z nakładek ładuje do zmiennej FRAMES określoną wartość, która natychmiast jest inkrementowana. Jeżeli program po załadowaniu samoczynnie startuje, to można oszacować wartość, jaką będzie miała zmienna FRAME w określonym punkcie programu zasadniczego. Tak więc w pewnym momencie realizacji programu (zazwyczaj blisko początku) wartość FRAME porównywana jest ze wzorcem, który może być pobrany z dowolnego miejsca pamięci. Jeżeli nie jest ona prawidłowa, program przechodzi do procedury zerowania systemu. Wystarczy sprawdzić tylko jeden z bajtów FRAME.

Złamanie opisanego zabezpieczenia może polegać na takim przerobieniu zapisanego w języku BASIC programu ładującego, aby wpisywał on również odpowiednią wartość do zmiennej FRAME. Bardziej eleganckim rozwiązaniem jest rozszyfrowanie (z pomocą disasemblera) głównego bloku programu i wyzerowanie (wpisanie rozkazów NOP — kod 00) fragmentu, który wykonuje sprawdzanie wartości FRAMES. Obydwa wymienione sposoby są możliwe dzięki informacjom zawartym w nagłówkach. Dlatego też inwencja producentów oprogramowania poszła w kierunku opracowania metody zapisywania na taśmie bloków, które nie zawierałyby nagłówków.

Zapisywanie bloków informacji nie poprzedzonych nagłówkiem wymaga programowania w kodzie maszynowym, gdyż zastosowane w ZX SPECTRUM instrukcje współpracy z magnetofonem zawsze powodują zapisanie (lub odczyt) nagłówka. Można się przy tym posłużyć procedurami zawartymi w firmowym ROM-ie.

Dla zapisania bloku informacji nie poprzedzonego nagłówkiem, można wykorzystać procedurę rozpoczynającą się od adresu $4C2_H$ (1218 dziesiętnie). Przed jej wywołaniem należy do akumulatora mikroprocesora Z80 wpisać liczbę, w której najbardziej znaczący bit będzie równy 1 (np. $FF_H$). W rejestrze IX należy umieścić adres początku bloku z ko-

dem maszynowym, który ma być zapisany na taśmie. Natomiast w parze rejestrów DE należy podać liczbę bajtów w bloku.

Przed blokiem informacji przez ok. 2 sekundy zapisywany jest ciąg impulsów o długości 2168 okresów zegara procesora (T). Pozwala on na odpowiednie ustawienie układów automatyki w magnetofonie, a przy odczytywaniu informuje układy wejściowe mikrokomputera, że za moment rozpocznie się informacja cyfrowa. Na końcu ciągu zapisywane są dwa impulsy synchronizujące: przez 667 okresów T wyjście „mikrofon" jest wyłączone, a następnie przez 735 okresów T jest ono włączone. Jako pierwszy bajt informacji zapisywany jest znacznik określający, czy dany blok jest nagłówkiem (wartość 00) czy blokiem danych (wartość $FF_H$). Na końcu bloku zapisywany jest bajt kontrolny. Powstaje on przez wykonywanie rozkazu XOR kolejnych bajtów danych z bajtem kontrolnym. Jako wartość początkową bajtu kontrolnego przyjmuje się 00 lub $FF_H$ — odpowiednio do wartości znacznika typu bloku.

Bloki bez nagłówka muszą być wczytywane przez procedurę napisaną w języku maszynowym. Można w tym celu wykorzystać procedurę umieszczoną w ROM-ie od adresu $556_H$ (1366 dziesiętnie). Przed jej wywołaniem należy do akumulatora wpisać znacznik typu $FF_H$ (nagłówek miałby wartość 00). W rejestrze IX należy podać adres, od którego dane mają być ładowane do pamięci. W parze rejestrów DE podaje się liczbę bajtów w bloku, który będzie ładowany. Ważne jest też, aby znacznik przeniesienia (ang. CARRY FLAG) był w stanie logicznym 1. Dowolny program, który ma być ładowany do komputera przez „normalnego" użytkownika (a więc instrukcją LOAD) musi zawierać blok zapisany w języku BASIC, zawierający nietrudną do zlokalizowania procedurę w kodzie maszynowym lub przynajmniej jeden blok z programem w języku maszynowym poprzedzony nagłówkiem. Pozostaje więc skorzystać z disasemblera, aby dowiedzieć się, jakie wartości ładowane są do poszczególnych rejestrów mikroprocesora przed wywołaniem procedury wczytującej blok bez nagłówka.

Aby zabezpieczyć się przed „niepowołanym" dostępem do wspomnianego bloku z kodem maszynowym, wymyślono kolejną sztuczkę: samostartujący blok napisany w języku maszynowym. Jest to możliwe przez odpowiednie wykorzystanie zmiennych systemowych komputera. Zapisywany blok kodu powinien zawierać oprócz programu również przestrzeń zmiennych systemowych oraz przestrzeń, w której przechowywany jest kod pośredni programów w języku BASIC. Ponadto należy również zapisać obszar stosu, który zajmuje miejsce poniżej adresu pamiętanego w zmiennej RAMTOP. Tak zapisany kod spowoduje wznowienie realizacji programu od miejsca, w którym została ona przerwana wykonaniem instrukcji SAVE. Przykładowo — zapis samoczynnie startującego kodu maszynowego można wykonać w następujący sposób:

```
10 SAVE „NAME" CODE 23552,4000
20 RAND USR 27000
```

Adres 23552 ($5C00_H$) określa początek obszaru zmiennych systemowych. Załadowany w ten sposób kod rozpocznie pracę od realizacji instrukcji RAND USR.

Okazuje się, że nawet tak wymyślne zabezpieczenie nie stanowi przeszkody dla piratów. Jeżeli program nie jest zbyt długi, należy odpowiednio zredukować wartość zmiennej RAMTOP — tak, aby można było go załadować powyżej wskazywanej przez nią granicy. Uniemożliwia to samoczynny start i pozwala na wykonanie dowolnych czynności (z ponownym zapisaniem na taśmę włącznie). Jeśli jednak program jest na tyle obszerny, że zmiennej RAMTOP nie można nadać tak niewielkiej wartości, to sposobem na przerwanie realizacji programu jest spowodowanie generacji komunikatu o błędzie zaraz po załadowaniu programu. Powoduje to przejście od interpretera języka BASIC, co wystarcza, by móc zająć się wpisanym programem. Dla generacji błędu przy odczycie wystarczy podać zbyt dużą długość bloku. Generacja błędu nie może tu zostać zablokowana w omawiany wcześniej sposób (poprzez zmienne DF-SZ lub ERR-SP), gdyż zapis poprzez instrukcję SAVE generuje komunikat w dolnej części ekranu.

Pozostaje jednak jeszcze przypadek, gdy blok kodu wypełnia cały obszar pamięci RAM od adresu $4000_H$ do $FFFF_H$. Otóż powstały już programy ładujące, umieszczane... w obszarze zmiennych systemowych, które ładują nawet bloki bez nagłówka w cały obszar pamięci, ale z wyłączeniem miejsca, które same zajmują.

Na zakończenie dodajmy, że programy można też kopiować z magnetofonu na magnetofon, choć oczywiście nie dostarcza to takiej satysfakcji jak łamanie zabezpieczeń. Metoda ta wymaga jednak sprzętu wysokiej klasy o szerokim paśmie przenoszenia (przez obcięcie wyższych częstotliwości „prostokątne" impulsy cyfrowe ulegają zaokrągleniu). Ważna jest też w tym przypadku precyzyjnie dobrana szybkość przesuwu taśmy. Z marnym pasmem przenoszenia krajowych magnetofonów można sobie poradzić włączając między magnetofony układ formowania impulsów.

**AJP**

**LITERATURA**

[1] Logan I., O'Hara F.: The Complete SPECTRUM ROM disasembly. Melbourne House Publishers Ltd., 1983.

---

**Czytelnikom, którzy od tego numeru rozpoczęli stały kontakt z mikroKLANEM winni jesteśmy kilka wyjaśnień na temat poniższego cyklu. W Akademii mikroKLANU nie uczymy programowania od podstaw. Zainteresowanych wprowadzeniem w programowanie odsyłamy do instrukcji obsługi posiadanego komputera lub do biblioteki, gdzie z pewnością znajdą kilka książek o programowaniu w języku BASIC. Wielbicieli kursów w odcinkach odsyłamy do „Horyzontów Techniki", które podjęły próbę uporządkowania prowadzonego tam kursu.**

**W Akademii mikroKLANU staramy się przekazać Czytelnikom umiejętność eleganckiego programowania, czyli takiego, które pozwoli nam samym (i innym!) wielokrotnie wykorzystać pracę włożoną w napisanie programu.**

## Akademia mikroKLANU (4)

## Kursor

Podział programu na moduły bardzo ułatwia opracowanie całości, a w przypadku długich programów jest właściwie jedynym sposobem przetestowania. Krótki „programik" łatwo objąć całościowo — łatwiej więc wprowadzać poprawki i usuwać błędy.

Język BASIC nie bardzo nadaje się do strukturalizacji, Zobaczmy jednak, co można zrobić. Pierwszym krokiem było już umieszczanie komentarzy w innych liniach niż instrukcje. Ten sam sposób możemy zastosować do podziału programu na moduły: niech wejście i wyjście mają numery linii rzędu dziesiątek, procedura główna rzędu setek, a procedury poboczne niech zaczynają się od numerów będących wielokrotnościami tysięcy. Tak naprawdę ułatwienie pisania programu poprzez jego strukturalizację zauważymy jednak, gdy jakaś część programu będzie się powtarzała (np. obliczanie wartości skomplikowanego wyrażenia lub gdy stosujemy podobne procedury wejścia-wyjścia w kilku miejscach w programie). Wtedy nasze gotowe, przetestowane moduły, możemy bardzo łatwo umieścić w programie, a nawet wykorzystać w innych programach.

Nasz szkolny program IV jest taką właśnie gotową procedurą, służącą do przesuwania kursora po całym polu ekranu dla wskazania pewnego położenia. Procedura ta została napisana w języku BASIC dla komputera ZX SPECTRUM. Ewentualne zmiany procedury w celu przy-

stosowania jej do innych komputerów są bardzo proste i ograniczają się do zmiany wielkości obrazu (zmienne L oraz H oznaczające szerokość i wysokość pola) oraz do zastąpienia instrukcji INKEY$ odpowiednią instrukcją odczytującą wciśnięty klawisz (np. w komputerach COMMODORE taką instrukcją jest GET, przy czym należy użyć zmiennej łańcuchowej, tzn. GET B$ i porównywać wczytany znak z opisami umieszczonymi na klawiszach). Należy też oczywiście zmienić początek pamięci obrazowej, czyli zmienną TP.

A oto procedura:

```
LIST

1000 REM PROCEDURA SUWANIA KURSOREM (JASNY KWADRAT)
1010 LET TP=22528
1011 REM TP JEST POCZATKIEM PAMIECI VIDEO (LEWY GORNY ROG)
1020 LET L=32
1030 LET H=22
1031 REM L ORAZ H OZNACZAJA ILOSC KOLUMN I WIERSZY OBRAZU VIDEO
1040 LET XG=0
1050 LET YG=0
1051 REM XG ORAZ YG TO AKTUALNE POLOZENIE KURSORA
1060 LET A=PEEK TP+XG+YG*L
1061 REM A ZAPAMIETUJE ELEMENT,GDZIE POJAWIA SIE KURSOR
1070 POKE TP+XG+YG*L,134
1071 REM KOD 134 OZNACZA MIGAJACY KWADRAT
1080 LET XX=XG
1090 LET YY=YG
1091 REM ZAPAMIETANIE OSTATNIEGO POLOZENIA KURSORA
1100 IF INKEY$="" THEN GOTO 1100
1101 REM KOMPUTER CZEKA NA WCISNIECIE KLAWISZA
1110 IF INKEY$="8" THEN LET XG=XG+1-INT((XG+1)/L)*L
1120 IF INKEY$="5" THEN LET XG=XG-1+(1-SGN(XG))*L
1130 IF INKEY$="6" THEN LET YG=YG+1-INT((YG+1)/H)*H
1140 IF INKEY$="7" THEN LET YG=YG-1+(1-SGN(YG))*H
1141 REM PRZESUNIECIA KURSORA ODPOWIADAJA NATURALNYM KLAWISZOM SPECTRUM
1150 IF INKEY$="Y" THEN RETURN
1151 REM WCISNIECIE RETURN (LUB Y) POWODUJE POWROT DO GLOWNEGO PROGRAMU
1160 POKE TP+XX+YY*L,A
1161 REM WYPELNIENIE POPRZEDNIEGO POLOZENIA KURSORA STARYM ELEMENTEM
1170 GOTO 1060
READY.
```

Jeszcze pewna uwaga praktyczna, ułatwiająca eleganckie pisanie programów. Kiedy ostatni raz Czytelnicy użyli funkcji SGN? Jest to standardowa funkcja w języku BASIC i jej wykonanie jest bardzo szybkie. W naszym szkolnym programie (linie 1120 oraz 1140) dzięki SGN udało się uniknąć skoków warunkowych — można powiedzieć, że funkcja SGN działa tu jako element decyzyjny. Takie sztuczki, choć nieco zaciemniają algorytm programu, ale za to r z e c z y w i ś c i e go skracają. Trzeba tylko je dobrze udokumentować!

Zachęcamy Czytelników do poeksperymentowania — komputera n i e m o ż n a zepsuć z klawiatury... Warto sprawdzić, co się stanie, gdy opuścimy linijkę 1160. A co będzie, jeśli opuścimy nasz trick w linii 1120?

**JAKUB TATARKIEWICZ**

---

- „Prace" nad powołaniem osobnego pisma „mikroKLAN" w toku, chociaż:
  - — brakuje papieru
  - — drukarnie robią bokami i bez mikroKLANU
  - — decydenci nie chcą słyszeć o wydawaniu zezwoleń na nowe czasopismo bo... obowiązuje moda na książki
  - — wydawnictwo nie może dać nowych etatów ze względu na złowieszczy FAZ. Są jednak ciągle tacy co wierzą, że pismo powstanie.
- Zniesienie bariery celnej na mikrokomputery przyniosło spodziewany raptowny spadek cen wolnorynkowych! ZX SPECTRUM 48K można było dostać za 100 ... 120 tys. zł., a COMMODORE C64 za ok. 160 tys. zł. Niektórzy pozbywali się komputerów za „zielone", w kwotach zbliżonych do obowiązujących na Zachodzie. Sytuacja zapewne nieco zmieniła się od 1 kwietnia, tzn. po zablokowaniu nieudokumentowanych wpłat na konta dewizowe typu A.

---

prowadzi:
Andrzej J. Piotrowski
tel. dom.: 48-22-85


## PRZEDSIĘBIORSTWO ZAGRANICZNE W POLSCE
## STARCOMP
### oferuje modułowy system mikroprocesorowy w postaci:

1. Jednostka centralna z mikroprocesorem INTEL 8080 (8253, 8259)
2. Jednostka centralna z mikroprocesorem Z 80 (8214, Z 80 CTC)
3. Pamięć 4 KB RAM (2114), 16 KB EPROM (2716)
4. Pamięć 16 KB RAM (6116), 32 KB EPROM (2764)
5. Pamięć 64 DRAM (4164), 32 KB EPROM (2764) — Memory Management
6. Sterownik monitora alfanumerycznego 64 znaki × 24 wiersze
7. Sterownik monitora graficznego 384 × 256
8. Pakiet wejść-wyjść równoległych (2 × 8255) wraz z programatorem EPROM (2716, 2732, 2764, 27128)
9. Pakiet wejść-wyjść szeregowych (2 × 8251, V-24, pamięć kasetowa)
10. Pakiety kontrolno-sterujące i serwisowe

System dostarczany jest w pełnym składzie, zestawiony w typowym stelażu przystosowanym do pakietów o wymiarach 140 × 150 mm, ze złączem 84 stykowym ELTRA CANNON, lub w postaci poszczególnych pakietów wybranych przez klienta. Na życzenie może być dostarczone oprogramowanie.

### PZ STARCOMP oferuje również:

11. Przełączniki typu X systemów MERA 9150 (SEECHECK)
12. Pakiety do pamięci dynamicznych o pojemności modułowej po 64 K słów 24-bitowych z kontrolą parzystości
13. Pamięci operacyjne do komputera ODRA 1305 oraz RIAD 32, zestawione z pakietów wymienionych w punkcie 12

**Wszelkich informacji udziela Biuro PZ STARCOMP, 01-548 Warszawa, ul. Czarnieckiego 64/2, tel. 39-22-91.**

*EO/436/K/85*

**Nowej wersji systemu CP/M nie wróży się na Zachodzie długiego życia, podobnie jak mikrokomputerom wykorzystującym mikroprocesory 8-bitowe. Z zachodniej perspektywy jest to jakby proteza — pozwalająca sprostać rosnącym wymaganiom użytkowników i przetrwać do momentu, gdy „szesnastki" staniejąa do kilku dolarów.**

**My mamy jednak inną perspektywę i — poza nielicznymi próbami składania IBM PC przez firmy polonijne — nic nie wróży rychłego upowszechnienia się w kraju mikrokomputerów 16-bitowych. CP/M 3+ jest więc może dla nas deską ratunku. Przekonstruowanie oferowanych w kraju mikrokomputerów pod kątem nowego systemu nie jest takie trudne. Jeśli mikrokomputery z serii MSX staniejąa na tyle, że odniosą wśród osób prywatnych podobny sukces jak ZX SPECTRUM, CP/M 3+ miałby na naszym rynku sporą szansę.**

# CP/M 3+

Najpopularniejszym systemem operacyjnym dla mikrokomputerów 8-bitowych (wykorzystujących Z80 lub 8080) jest system CP/M — wersja 2.2. Dysponuje on niedużą pamięcią operacyjną (maks. 64 KB), co w poważnym stopniu ogranicza jego możliwości. Obecnie firma DIGITAL RESEARCH opracowała nową wersję systemu, oznaczoną **CP/M — Plus wersja 3.0**, która może zarządzać znacznie większym obszarem pamięci operacyjnej. Osiągnięto to nie tylko na drodze rozwiązań programowych, lecz przede wszystkim — przez zastosowanie nowego sposobu połączenia bloków pamięci operacyjnej.

Możliwa jest dzięki temu całkowita zmiana koncepcji pracy systemu, co czterokrotnie zwiększa jego wydajność. System jest w stanie obsługiwać 16 jednostek pamięci na dyskach elastycznych. Ponadto wprowadzono wiele rozwiązań ułatwiających eksploatację systemu.

Dodatkową zaletą systemu CP/M-Plus jest możliwość wykonywania pod jego nadzorem prawie wszystkich programów działających pod systemem CP/M-2.2. Wyjątek stanowią programy odwołujące się bezpośrednio do procedur w module BIOS (Basic Input Output System) wersji 2.2.

## Organizacja pamięci operacyjnej

Możliwość bezpośredniego adresowania pamięci przez procesor 8-bitowy jest ograniczona do 64 KB. Limituje to maksymalną długość segmentów programów użytkowych, liczbę obsługiwanych urządzeń we-wy, szybkość działania, itp.

Projektanci systemu CP/M-Plus 3.0 znaleźli takie rozwiązanie problemu, które po stronie sprzętu (ang. hardware) nie wymaga skomplikowanych konstrukcji. Polega ono na zastosowaniu specjalnego układu, pozwalającego na równoległe [1]) połączenie ze sobą wielu bloków pamięci operacyjnej. W ten sposób można uzyskać pamięć operacyjną złożoną z 16 bloków po 64 KB każdy, co daje łączny obszar o wielkości 1 MB.

Minimalny obszar pamięci operacyjnej, umożliwiający korzystanie ze wszystkich funkcji systemu operacyjnego, wynosi 128 KB (dwa bloki). Możliwa jest także instalacja systemu w pamięci 64 KB, wiąże się to jednak z rezygnacją z wielu funkcji oraz znacznym obniżeniem wydajności.

Z każdego z połączonych bloków pamięci wydzielony zostaje obszar o wielkości 4 do 16 KB, przez co uzyskuje się jeden obszar, traktowany przez system jako spójny. Obszar ten jest wykorzystywany w procesie zarządzania pamięcią operacyjną oraz pośredniczy przy realizacji odwołań programów do systemu operacyjnego.

Kompletny system zajmuje 21 KB w pierwszym bloku pamięci. Obszar przeznaczony na wczytywanie programów jest wydzielony z 1 i 2 bloku i wynosi 60 KB. Istnienie pozostałych (do 14) bloków pamięci operacyjnej uzależnione jest od liczby podłączonych jednostek dyskowych oraz od planowanej wydajności systemu. Bloki te są wykorzystywane wyłącznie przy obsłudze jednostek dyskowych.

## Obsługa pamięci dyskowych

Głównym czynnikiem obniżającym szybkość działania systemów mikrokomputerowych jest współpraca z jednostkami sterującymi dysków elastycznych. Straty czasowe rosną wraz z liczbą dostępów do tych pamięci. W systemie CP/M-Plus problem ten rozwiązywany jest różnymi metodami, sprowadzającymi się jednak do utrzymania liczby dostępów na jak najniższym poziomie. Odbywa się to kosztem poświęcenia przeważającej części pamięci operacyjnej na obsługę pamięci dyskowych.

Jedna z zastosowanych metod polega na wczytywaniu do pamięci operacyjnej całej tablicy adresowej dyskietki (jej długość wynosi często wiele KB). W ten sposób na przeszukanie całej tablicy wymagany jest tylko jeden dostęp do dyskietki. Metoda ta jest określana w języku angielskim jako Directory Caching.

Druga metoda polega na przypisaniu dużych obszarów pamięci operacyjnej dla poszczególnych jednostek dyskowych na przetwarzanie danych pobieranych z tych jednostek (ang. cache memory). Umożliwia to operowanie na dużych częściach zbiorów zamiast na pojedynczych rekordach, co także obniża liczbę dostępów do dyskietek. W momencie gdy program żąda dostępu do określonego rekordu, system operacyjny sprawdza w pierwszej kolejności, czy rekord ten znajduje się już w pamięci operacyjnej. Dopiero gdy go tam nie znajdzie, następuje pobranie z dyskietki odpowiedniej części zbioru.

Jak widać, system CP/M-Plus kieruje się zasadą, że w trakcie jednego dostępu należy czytać i zapisywać w pamięci dyskowej jak najwięcej danych. Zasada ta jest stosowana także w przypadku tworzenia nowego zbioru. Rekordy tworzone w pamięci operacyjnej przepisywane są na dyskietki dopiero po zapełnieniu się obszaru przeznaczonego na ten cel. Maksymalna liczba danych przepisywanych w ten sposób wynosi 16 KB. Podobna transmisja wymagałaby w systemie CP/M-2.2 128 dostępów do pamięci dyskowej.

## Dodatkowe usprawnienia systemu

W porównaniu do wersji 2.2, w systemie CP/M-Plus 3.0 wprowadzono ok. 130 nowych funkcji — ułatwiających jego użytkowanie.

Między innymi zautomatyzowano procedurę zmiany dyskietek. System sam rozpoznaje pojawienie się nowej dyskietki i odnotowuje ten fakt w odpowiedniej tablicy, nie czekając na dyrektywę operatorską.

Innowacją jest także wprowadzenie hasłowej ochrony całych dyskietek lub poszczególnych zbiorów. Ochronie może podlegać każda z podstawowych operacji przeprowadzanych na zbiorach, tzn.: zapis, odczyt i zamazywanie informacji. Wprowadzono także wiele dodatkowych funkcji ułatwiających obsługę zbiorów dyskowych.

Rozbudowano również obsługę błędów przez wprowadzenie wyczerpujących komunikatów o błędach. Umożliwiono korygowanie niektórych błędów w drodze dialogu z systemem operacyjnym (pytanie systemu, odpowiedź użytkownika). Dopiero przy braku możliwości bieżącej korekcji błędu następuje całkowite przerwanie wykonywania programu.

Na uwagę zasługuje także fakt zwiększenia liczby programów usługowych (ang. utilities) działających w systemie. Aby uprościć obsługę tych programów, wprowadzono funkcję HELP, która umożliwia korzystanie z nich bez potrzeby używania podręcznika.

**WITOLD WOŹNIAK**
Warszawa

[1]) Procesor zawsze „widzi" przestrzeń 64KB. Rozwiązanie polega na przyłączaniu do procesora (pod kontrolą systemu) żądanego w danym momencie bloku — przyp. AJP

**Wśród opisywanych w mikroKLANIE bloków funkcjonalnych wchodzących w skład mikrokomputera brakowało do tej pory rozwiązań umożliwiających wyświetlanie informacji na ekranie telewizyjnym. Wypełniając tę lukę, rozpoczynamy cykl publikacji o sterownikach CRT. Zaczynamy od rozwiązania uproszczonego, zrealizowanego na łatwo dostępnych elementach — układach TTL.**

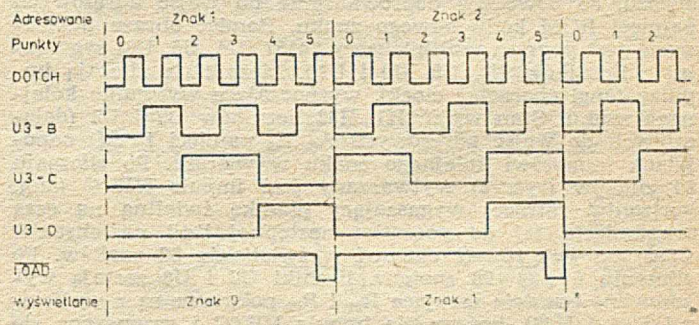
**Prezentowane rozwiązanie umożliwia zapisywanie do 64 znaków w wierszu (więcej niż wiele popularnych komputerów domowych). Chcąc zapisać większą liczbę — trzeba by wykorzystać profesjonalny monitor ekranowy — natomiast opisaną konstrukcję można podłączyć do nieznacznie tylko zmodyfikowanego telewizora (dobudowa wejścia video). Warto jeszcze dodać, że opisane rozwiązanie można w przyszłości zmodyfikować (rozbudowując generator znaków o pamięć RAM) — tak, aby możliwe było wykorzystywanie pseudografiki.**

# Alfanumeryczny sterownik monitora telewizyjnego (1)

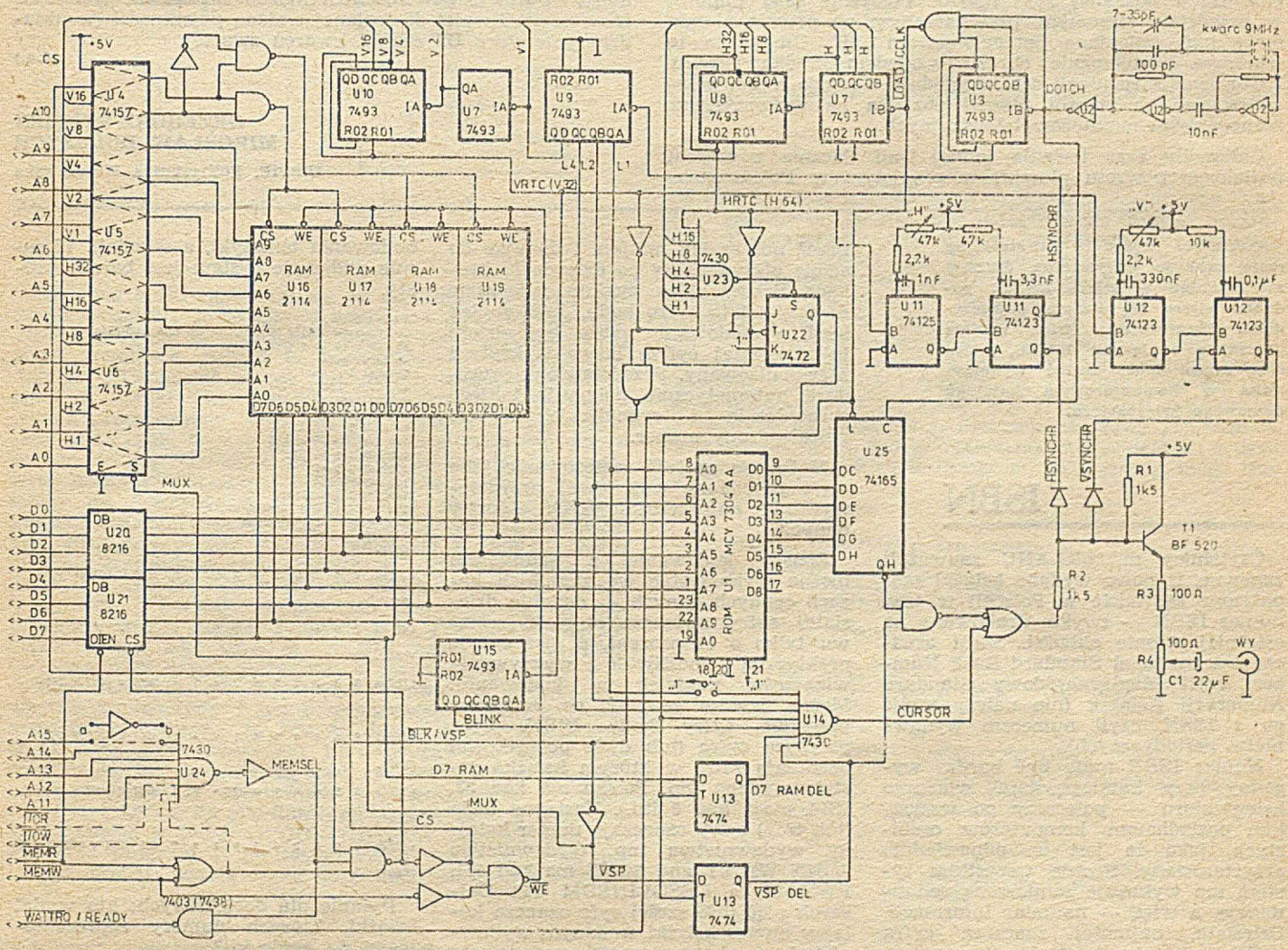
Opisany w tekście sterownik (rys. 1) umożliwia wyświetlanie na ekranie monitora telewizyjnego 2048 znaków alfanumerycznych, rozmieszczonych w 32 wierszach o 64 znakach. Wyświetlane na ekranie znaki wpisywane są przez mikroprocesor do wydzielonego zespołu 2 KB pamięci RAM, stanowiącego pamięć obrazu. Adresy poszczególnych bajtów pamięci obrazu odpowiadają określonym miejscom na ekranie — tzn. numerom kolumn i wierszy. Znaki zapisywane są w 7-bitowym kodzie ASCII. Ósmy bit (nie wykorzystywany w tym kodzie) stosowany jest do włączania kursora.

Znaki wyświetlane są na ekranie monitora przez rozjaśnienie odpowiednich punktów w polu obejmującym osiem kolejnych linii po sześć punktów każdej.

Zasadniczą funkcję przetwarzania zawartości pamięci obrazu na sygnały rozświetlające punkty ekranu realizuje specjalny układ pamięci stałej ROM — tzw. generator znaków (U1). Na wejścia adresowe $A_3$...$A_9$ tego układu podawany jest z pamięci obrazu kod wyświetlanego znaku. Na




**Rys. 2. Przebiegi czasowe licznika punktów U3**

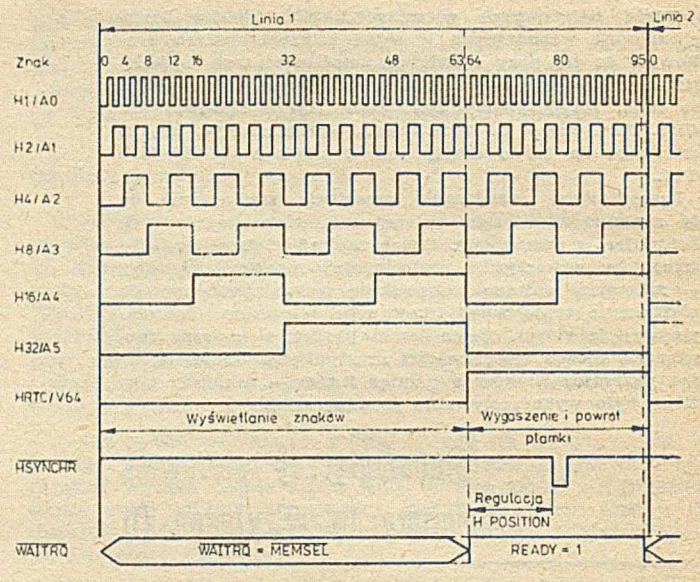



**Rys. 1. Schemat ideowy sterownika monitora telewizyjnego**

wejścia adresowe $A_0...A_2$ podawany jest numer wyświetlanej linii danego znaku. Sygnały na wyjściach $D_0...D_7$ odpowiadają rozkładowi punktów świetlnych w aktualnie wyprowadzanej, linii znaku. Sygnały te są podawane kolejno (szeregowo) do wyjścia wizyjnego przez rejestr przesuwny 74165 (U25). Wyprowadzanie sygnałów jest taktowane przez tzw. generator punktów (jego częstotliwość odpowiada czasowi wyświetlania punktu na ekranie). Dane z pamięci ROM wpisywane są do rejestru przesuwnego w trakcie trwania impulsu LOAD/-CCLK z licznika U3, odliczającego sześć punktów (jedna linia znaku). Impuls ten formowany jest przez współpracującą z licznikiem U3 bramkę (rys. 2).

Przejście do wyświetlania kolejnego znaku (ta sama linia) wymaga powiększenia o 1 adresu podawanego do pamięci obrazu, co powoduje doprowadzenie do wejść adresowych pamięci ROM kodu nowego znaku. Pamięć obrazu adresowana jest poprzez multipleksery 74157 (U4, U5, U6) sygnałami z liczników U7, U8, U9 i U10. Liczniki te zliczają impulsy (ang. character clock), wyznaczające wysyłanie kolejnego znaku. Stan wyjść H1...H32 liczników U7 i U8 (dołączonych do wejść adresowych $A_0...A_5$ pamięci RAM) odpowiada numerowi kolejnego znaku w wierszu. Po zliczeniu 64 znaków (rys. 3) wytwarzany jest impuls HRTC (ang. horizontal retrace) wygaszający plamkę świetlną na czas przesunięcia jej do początku następnej linii na ekranie. Czas ten jest równy czasowi wyświetlania 32 znaków. Po zliczeniu 64+32=96 znaków, liczniki U7 i U8 zostają wyzerowane (wejścia zerujące $R_{01}$ i $R_{02}$ połączone są z wyjściami H64 i H32), kończy się impuls HRTC i rozpoczyna się liczenie znaków następnej linii obrazu. Impuls HRTC wyzwala uniwibratory U11 (74123), wytwarzające impuls synchronizacji poziomej HSYNCHR/. Położenie tego impulsu w przedziale czasowym, przeznaczonym na powrót plamki może być zmieniane potencjometrem „H". Umożliwia to właściwe usytuowanie obrazu względem środka ekranu. Zastosowanie regulacji oraz względnie długiego czasu, przeznaczonego na powrót plamki pozwala na wykorzystanie różnego typu monitorów telewizyjnych.

Całkowity czas trwania jednej linii (łącznie z czasem poziomego powrotu plamki) w przyjętym w Polsce standardzie telewizyjnym jest równy 64 µs. Częstotliwość generatora punktów U2 powinna wynosić zatem 6×(64+32):64 µs=9 MHz. Częstotliwość ta może różnić się od wymaganej o ok. ±10%. W przypadku trudności z uzyskaniem odpowiedniego kwarcu z powodzeniem można zastosować generator RC o częstotliwości dostrajanej zmianą pojemności (w układzie generatora U2 zaciski kwarcu zwarte).




**Rys. 3. Przebiegi czasowe liczników znaków**

(c.d.n.)

**GRZEGORZ STĘPIEŃ**
**MIROSŁAW DOLEŻYCH**
**IBSPiE, Politechnika Warszawska**

---

**Króciutkie procedury (niestety na razie brakuje miejsca na dłuższe) zamieszczane w mikroKLANIE podobno cieszą się dużym powodzeniem. Prezentujemy więc kolejną procedurę w języku BASIC, która tym razem może zostać wykorzystana w komputerowym katalogu domowego księgozbioru.**

# ISBN

Czytelnicy mikroKLANU zauważyli zapewne, iż nowo wydane książki zaopatrzone są (także w Polsce!) w tzw. numer ISBN — zwykle drukowany na ostatniej stronie okładki. Skrót oznacza International Standard Book Number, czyli Międzynarodowy Standard Numeracji Książek (nie należy mylić go z ISSN, czyli numerem wydawnictw periodycznych).

Numer ISBN może być bardzo wygodny przy katalogowaniu własnego księgozbioru — polecamy go wszystkim posiadaczom komputerów osobistych, jako że jest on odpowiednio przystosowany. Przede wszystkim system ten wykazuje zamierzoną redundancję, a więc — nadmiar informacji. Dziesiąta, najmniej znacząca cyfra ISBN jest cyfrą kontrolną (sprawdzającą): mnożąc kolejne cyfry ISBN przez numer ich miejsca i dodając do siebie otrzymujemy po dziewięciu krokach liczbę, która modulo jedenaście daje dziesiątą cyfrę kodu. Jakikolwiek błąd w kodowaniu może być dzięki tej dziesiątej cyfrze łatwo wykryty i zasygnalizowany. Przedstawiona procedura automatycznie dokonuje takiego sprawdzenia i sygnalizuje błędnie wprowadzony numer.

Może ona znaleźć zastosowanie jako część procedury wejściowej przygotowywania najprzeróżniejszych zbiorów danych.

Sposób kodowania z nadmiarem można wykorzystać we wszelkich bazach cennych danych — dodanie dziesiątej cyfry zwiększa objętość zestawu tylko o 11 procent, a gwarantuje praktyczną możliwość wykrywania większości pomyłek w kodowaniu. Można jeszcze zapytać, co oznaczają pozostałe cyfry kodu ISBN. Otóż pierwsza cyfra (lub dwie początkowe) oznaczają kraj, w którym książka została wydana (np. Polska ma kod 83, USA mają kod 0 itd.). Następne dwie (lub trzy) cyfry oznaczają numer kolejny wydawnictwa (np. wydawnictwo JOHN WILEY and SONS ma kod 471, PWN 01, a OSSOLINEUM 04). Dalszych pięć (lub sześć) cyfr oznacza kolejny numer książki w danym wydawnictwie i niczym specjalnym się nie wyróżnia (nie ma tu możliwości zakodowania tematyki, a szkoda!). W sumie widać, że system jest bardzo prosty.

```
10 DIM A(10)
20 PRINT "WPROWADZ NUMER ISBN (OD LEWEJ DO PRAWEJ)"
30 FOR I=10 TO 1 STEP -1
40 INPUT A(I)
50 NEXT I
60 PRINT "LICZBA ";
70 FOR I=10 TO 1 STEP -1
80 PRINT A(I);
90 NEXT I
100 S=0
110 FOR I=1 TO 10
120 S=S+A(I)*I
130 NEXT I
140 S=S-INT(S/11)*11
150 IF S<>0 THEN PRINT " NIE";
160 PRINT " JEST W KODZIE ISBN"
170 END
READY.

LICZBA 0 8 3 7 0 6 4 2 6 1  JEST W KODZIE ISBN

READY.

LICZBA 2 0 8 4 3 1 0 8 1 0  JEST W KODZIE ISBN

READY.

LICZBA 8 3 7 1 5 8 5 6 5 4  JEST W KODZIE ISBN

READY.

LICZBA 9 6 3 0 5 3 1 9 4 1  JEST W KODZIE ISBN

READY.

LICZBA 3 5 4 0 1 0 7 4 9 5  JEST W KODZIE ISBN

READY.
 5
LICZBA 5 0 0 2 0 0 0 1 9 5  NIE JEST W KODZIE ISBN

READY.
```

Pytanie dla dociekliwych: kto wydał książki, których numery zaprezentowano na wydruku?

**JAKUB TATARKIEWICZ**